BAKER BOTTS L.L.P.

30 ROCKEFELLER PLAZA

NEW YORK, NEW YORK 10112-0228

TO WHOM IT MAY CONCERN:

Be it known that WE, Armin Amrhein, Johannes Birzer, Thomas Hennefelder, Martin

Kiesel, Raimund Kram and Regina Schmitt, citizens of Germany, residing in Kuemmersbruck,

Stulln, Sugenheim, Poxdorf, Erlangen and Erlangen respectively, whose post office addresses are

Dresdnerstr. 16, 92245 Kuemmerstruck, Germany; Friedhofweg 2, 92551 Stulln, Germany;

Deutenheim 35, 91484 Sugenheim, Germany; Jahnstr. 36, 91099 Poxdorf, Germany; Fliederstr.

7A, 91056 Erlangen, Germany; and Herbstaeckerweg 5, 91056 Erlangen, Germany; respectively,

have invented an improvement in:

PROGRAMMING OF CYCLICAL MACHINES

of which the following is a

SPECIFICATION

BACKGROUND OF THE INVENTION

[0001]   The invention relates to a method of programming cyclical machines, in particular

production machines.

[0002]   It is customary nowadays, both for stored-program control (SPC) and for motion

control (MC), to model hierarchical running levels that are different in each case and are

assigned software tasks for controlling the respective technical process. These tasks may

perform system functions, but may also be user-programmed.

[0003]   It is known from DE 197 40 550 A1 that process control functionalities of the stored-program controllers "SPC" and motion functionalities of MC controllers can be integrated in a uniform configurable control system.

[0004]   This SPC/MC integration takes place in the form of interconnecting SPC and MC control modules. However, when the integration is carried out in such a way, an optimum and efficient task structure is not achieved for the entirety of the control tasks. Furthermore, with this type of integration it is mainly the classic MC functionalities, as are relevant in particular for machine tools, that are supported. Requirements for the controller, as they are known from the operation of production machines, are not optimally supported by this type of interconnection of SPC and MC control modules.

[0005]   It is known from EP 0 735 445 A2 to use a separate waiting command (WAIT) for the operation of a machine tool or a robot. However, the waiting command (WAIT) described here still does not optimally support the control of production machines in particular.

[0006]   In the application DE 19 93 19 33.2 it is proposed to use the clock of the communication system between the PC system and the peripheral devices for a change between a real-time operating program and a non-real-time operating program. Here, however, it is the task of this clock pickup from the communication system to allow the smoothest possible change to take place between real-time and non-real-time applications in an industrial process. In this configuration, the basic clock is only derived however from the clock of the communication medium and it is only used for the changing of the operating system mode of a PC system.

[0007]   Furthermore, the methods customary nowadays for the programming of cyclical

machines require different programs, the coordination of which is carried out by means of what are known as "event handlers". The programming of cyclical machines is consequently laborious and an "event handler" causes overheads and affects performance. Cyclical machines in the steady state are distinguished by a periodically recurring process or cycle. The productivity of the machine is directly dependent on this cycle. A shortening or reduction of the cycle enhances the productivity and the performance of the machine. Examples of cyclical machines are: packaging machines (for example tubular bag machines, filling machines or sealing machines) but also presses (for example roll-feed machines). For the performance of packaging machines, rapid printed mark synchronization and an associated rapid superposing of corrective movements are decisive for example. Such objects are still not optimally achieved today.

[0008]　The invention is therefore based on the object of creating in a simple manner a configurable running level model for the control tasks of an industrial controller for cyclical machines for in each case different control tasks and different boundary conditions or requirements of the underlying technical process, it being possible for the programming of the machine sequence to take place in a sequential program.

[0009]　The inventors make the assumption here that this is achieved in principle on the one hand by a uniform configurable running level model for the control tasks of the industrial controller and on the other hand by mechanisms (wait_for_condition command) which enable a user to wait for any desired conditions and respond with higher priority in the program flow.

[0010]　Setting out from this approach, the object stated above is achieved by the following

successive steps:

    a) creating an industrial controller equipped with a runtime system and having prioritized running levels and tasks, at least one sequential running level being created;

    b) formulating a machine sequence in a sequential program; and

    c) in the sequential program, providing by specific mechanisms that the waiting for conditions to be satisfied can be carried out with high priority and after the condition has been satisfied, the subsequent program sequence can be carried out with high priority up to a defined user-programmed end, the running levels being assigned system and/or user programs.

[0011]   The prioritizing of the running levels allows the tasks for operating the controller to run on levels of different priority according to their importance or the requirements which they have to meet (for example with respect to the response time behavior). A further advantage is that the user can formulate complex synchronization conditions (for example waiting for the occurrence of conditions) in a sequential program by simple mechanisms, without further mechanisms, such as for example event handlers, being required. This on the one hand avoids a management overhead in the controller and on the other hand supports the locality principle from a programming viewpoint. The fact that programs can be additionally loaded into the running levels allows the user to adapt the functionality of the controller very flexibly to the underlying requirements of the technical process.

[0012]   The mechanism described and the associated command are referred to in the exemplary embodiment as the "wait_for_condition".

[0013] A first refinement of the present invention is that the running levels are created from system levels and/or user levels. As a result, the user has the possibility of creating very flexibly a runtime system for an industrial controller which is appropriate for the respective requirements and boundary conditions of the technical process.

[0014] A further refinement of the present invention is that the running level model is clocked, the basic clock being derived from an internal timer or from an internal clock of a communication medium or from an external device or from a variable which belongs to the technological process. As a result, the basic clock for the running level model can be derived in a very flexible and very easy manner. The fact that the basic clock for the running level model can also be derived from a variable which belongs to the technological process allows direct feedback from the technological process to the controller to be obtained in a very easy way.

[0015] An exemplary embodiment of the invention is explained below and represented in the drawing, in which:

Figure 1 shows the main running levels of a classic stored-program controller;

Figure 2 shows the main running levels of a motion controller;

Figure 3 shows a schematic representation of an industrial controller;

Figure 4 shows the running level model of the industrial controller according to the invention;

Figure 5 shows an exemplary embodiment of the loading of user programs into the user levels;

Figure 6 shows an exemplary embodiment of the use and mechanism of the

wait_for_condition command in the running level model of the industrial

controller according to the invention;

Figure 7 shows a further exemplary embodiment of the use and mechanism of the

wait_for_condition command in the running level model of the industrial

controller according to the invention;

Figure 8 shows the syntactic description of the wait_for_condition command in a syntax

diagram;

Figure 9 shows an example of the formulation of an expression in programming-language

notation; and

Figure 10 shows in a schematic representation possibilities for how the basic clock is

obtained for the industrial controller.

[0016]   In the representation according to Figure 1, the main running levels of a classic stored-

program controller (SPC), arranged according to their priority, are shown. The increase in

priority is symbolized there by an arrow. In the lowest-priority level, as indicated by a dashed

line, two different tasks are performed, to be specific a free cycle, i.e. "user level free cycle", and

a background system level, i.e. "system level background". The background system level is

assigned, for example, communication tasks. In a following user level, referred to as "user level

time-controlled", the calling clock of the tasks or of the programs of this level can be

parameterized. Monitoring takes place to ascertain whether the processing of a user program of

this clocked level has been completed in time before the start event occurs once again. If the

clock time elapses without the user program of the assigned level being processed to completion,

a corresponding task of a next-but-one, in priority terms, "user level for asynchronous faults" is

started. In this "user level for asynchronous faults", the user can program out the handling of fault states.

[0017]   The "user level time-controlled" is followed by a "user level events". The response to external or internal events takes place within the "user level events". A typical example of such an event is the switching of a binary or digital input, whereby typically an event is triggered. In a "system level high priority" lie the tasks of the operating system which ensure the operating mode of the programmable controller (SPC).

[0018]   The representation according to Figure 2 shows the main running levels of a motion controller (MC). Here, too, the individual levels are arranged hierarchically according to their priority, as symbolized by an arrow. A "system level background" and a "user level sequential" have an equal priority, that is the lowest priority. This unified nature in terms of tasks is symbolized as in figure 1 by a dashed line. The tasks of the "user level sequential" are processed together with the tasks of the "system level background" in the round-robin procedure. Typical tasks of the "system level background" are, for example, those for communication tasks. In the "user level sequential", the parts of the program programmed by the user run for the actual control task. If, in one of these parts of the program, the controller encounters a movement or positioning command, a "suspend" is set, i.e. the user program is interrupted at this point. In this case, a command is synchronously used. The processing of this movement or positioning command takes place in a highest-priority "system level clocked". Each and every position controller or interpolator which is running in the "system level clocked" executes this movement or positioning command. After execution of the command, control returns to the "user level

sequential" and the user program interrupted by "suspend" is continued by a "resume" at the same point. The "system level clocked" contains not only the already mentioned position controllers but also the interpolation part of the control.

[0019]   The "user level events" resumes at the lowest-priority level. Accommodated here are those tasks which respond to external or internal events. Such events may be alarms, for example.

[0020]   In a following "user level synchronously clocked", synchronously clocked user tasks are performed, for example controller functionalities. These tasks are synchronized in relation to clocked system functions, such as for example the interpolator, position controller or cyclical bus communication.

[0021]   In the representation according to figure 3, it is shown in the form of a structure diagram that the control of a technical process P1 is performed by means of the runtime system RTS of an industrial controller. The connection between the runtime system RTS of the controller and the technical process P1 takes place bidirectionally via the inputs/outputs EA. The programming of the controller, and consequently fixing the behavior of the runtime system RTS, takes place in the engineering system ES. The engineering system ES contains tools for configuring, project planning and programming for machines or for controlling technical processes. The programs created in the engineering system are transferred via the information path I into the runtime system RTS of the controller. With respect to its hardware equipment, an engineering system ES usually comprises a computer system with graphics screen (for example display), input aids (for example keyboard and mouse), processor, main memory and secondary

memory, a device for accepting computer-readable media (for example floppy disks, CDs) and also connection units for data exchange with other systems (for example further computer systems, controllers for technical processes) or media (for example the Internet). A controller usually comprises input and output units, and also a processor and program memory. It is also conceivable for the control of a technical process P1 to be performed by means of a plurality of runtime systems RTS of industrial controllers.

[0022]   The representation according to Figure 4 shows the running level model of the industrial controller. The prioritizing of the levels is indicated by an arrow in the direction of the highest priority. The lowest-priority levels are the "cyclical user level" and the "sequential user level". These two levels run with the same priority. Therefore, these levels are separated in the representation according to Figure 4 by a dashed line. The "cyclical user level" includes the "background task", which is cycle-time-monitored. In the "sequential user level", the "motion tasks" are run through. "Motion tasks" are not cycle-time-monitored and serve essentially for describing sequential sequences. "Motion tasks" are processed virtually in parallel. Generally, all the user levels contain one or more tasks. The tasks receive the user programs. The tasks of the "cyclical user level" and of the "sequential user level" are processed in a common round-robin cycle.

[0023]   The next-following level is the "time-controlled user level". The tasks of this level are activated in a time-controlled manner. The time control can be set in a granularity of milliseconds. The "time-controlled user level" is followed by the "event-controlled user level". In this level, after detection of a user interrupt, what are known as "user interrupt tasks" are

activated. User interrupt events may be formulated as a logical combination of process events and/or internal states.

[0024] The next-higher level is the "user level for system exceptions". In this "user level for system exceptions", monitoring is carried out of system interrupts, the occurrence of which has the effect of generating what are known as "exceptions", i.e. instances of handling exceptional cases. In the "user level for system exceptions" there are, for example, the following tasks, which are activated when a corresponding system interrupt occurs:

a) "time fault task", which is activated when time monitors respond;

b) "peripheral fault task", which is activated for example in the event of process and diagnosis alarms, but also in the event of station failure or station return;

c) "system fault task", which is activated in the event of general system faults;

d) "program fault task", which is activated in the event of programming faults (for example division by zero);

e) "time fault background task", which is activated when the cycle time monitoring of the background task responds; and

f) "technological fault task", which is activated in the event of technological faults.

[0025] Following next is the group of levels "synchronously clocked levels". This group of levels has the highest priority in the running level model. The individual levels of this group of levels may have further prioritizings with respect to one another. The group of levels "synchronously clocked levels" comprises at least one system level and at least one user level. The system levels include the system functions such as, for example, position controller or

interpolator. User programs (AP1 - AP4; Figure 5) can be flexibly loaded in addition by a user into the user levels of this group of levels.

[0026] For the clock control of the "synchronously clocked levels" there are a series of different possibilities for clock generation. The basic clock may come, for example, from an internal timer (T1; Figure 10) or from an internal clock (T3; Figure 10) of a communication medium (for example Profibus) or else the clock may also be derived from a process event of the technological process. Such a process event may be, for example, the clock rate (TG; Figure 10) of an operation on a production machine or packaging machine. User levels of the group of levels "synchronously clocked levels" may in this case be clocked on the basis of the basic clock, but they may also run synchronously in relation to one of the system levels of the group of levels "synchronously clocked levels". The user tasks of this user level synchronous to a system level consequently have a synchronous, i.e. deterministic, relationship with a system level which can be flexibly fixed by the user. This has the advantage that deterministic responses to system tasks (system tasks run in the system levels) which the user has programmed in his user tasks, which run in the user levels of the group of levels "synchronously clocked levels", are guaranteed by the system. That is to say, for example, that the system guarantees that this "synchronous user level" is correspondingly activated for example before the interpolator, or else before any other desired system function.

[0027] The "time-controlled user level", the "event-controlled user level", the "sequential user level", the "cyclical user level" and the "user level for system exceptions" are optional.

[0028] The task of the "cyclical user level" (background task) is cycle-time-monitored. The

"motion tasks", on the other hand, are not cycle-time-monitored and serve essentially for describing sequential sequences. That is to say the present running level model supports a user both in the programming of sequential sequences and in event programming. Consequently, synchronous events and asynchronous events can be covered by the programming. The user programs (AP1 - AP4; Figure 5) created by the user can be loaded in addition into the user levels. The user programs AP1 to AP4 are usually created with the aid of a programming environment of the engineering system (ES; Figure 3).

[0029]   The representation according to Figure 5 shows an exemplary embodiment of the additional loading of user programs into the user levels. Figure 5 shows by way of example distinctive characteristics of user levels of the running level model. It is shown by the three points at the lower edge of the drawing that there may also be still further user levels, or else system levels. The prioritizing of the levels is indicated as above by an arrow in the direction of the highest priority. The user levels are assigned the user programs AP1 to AP4, indicated at the right-hand edge of the figure by small squares. The assignment is shown by assignment arrows ZP1 to ZP4. In the user levels there are tasks which receive the additionally loaded user programs AP1 to AP4. These tasks are then run through or processed in accordance with a specific strategy (for example sequentially). They may continue to have the property that they are run-time-monitored.

[0030]   The representation according to Figure 6 shows an exemplary embodiment of the use and mechanism of the wait_for_condition command, in the running level model of the industrial controller according to the invention. The wait_for_condition command (represented in figure 6

as wait_for_cond()) is used by way of example in this representation in the "sequential user level". The wait_for_condition command is used in the "motion tasks" MT1 and MT2 created by the user, which are a component part of the "sequential user level". The "motion tasks" MT1 and MT2 are in a round-robin cycle, represented by the arrow from MT1 to MT2 and by the angled-away arrow from MT2 to MT1. The three points inside this arrow indicate that there may be still further "motion tasks" in the round-robin cycle. The "motion task" MT1 contains the wait_for_condition command "wait_for_cond(cond_1)", the "motion task" MT2 contains the wait_for_condition command "wait_for_cond(cond_2)". Three points used in each case within MT1 and MT2 indicate that, in addition to the two wait_for_condition commands and the three positioning commands pos1() to pos3(), still further commands may be contained in the "motion tasks".

[0031]   Altogether, the running level model, represented by way of example in figure 6, of a runtime system for an industrial controller comprises the following levels (enumerated from the lowest to the highest priority): "cyclical user level", "sequential user level" (the tasks of these two levels have the same priority, represented by the dashed line between these levels), "time-controlled user level", "event-controlled user level", "user level for system exceptions", "synchronously clocked user level 2", "synchronously clocked user level 1", "synchronously clocked system level 2" and, as the highest-priority level, a "synchronously clocked system level 1".

[0032]   The operating mode of the wait_for_condition command is shown by way of example by "wait_for_cond(cond_1)" from the "motion task" MT1. If the "motion task" MT1 is next in

turn in the round-robin cycle, the commands of the "motion task" MT1 are serviced until the time slice has elapsed, or an interruption occurs. If this is the case, the "motion task" MT2 is serviced as the next task in the cycle, etc. If the wait_for_cond(cond_1) command is processed in the "motion task" MT1, the condition cond_1 is checked. If cond_1 = true, that is to say is satisfied, the next-following command pos2() is immediately executed and, if appropriate, further commands present in MT1 are successively processed, until control is passed to the next task.

[0033]   If the condition cond_1 = false, that is to say is not satisfied, the "motion task" MT1 is immediately interrupted and MT2 is serviced in the round-robin cycle. The condition cond_1 is inserted, however, into the "synchronously clocked system level 2" (indicated by the solid angled-away arrow from the wait_for_cond(cond_1) command to the "synchronously clocked system level 2") and is checked in the clock cycle of this system level to ascertain whether it has been satisfied. If the condition cond_1 is satisfied, the current task is displaced in the round-robin cycle, i.e. it has the time slice withdrawn from it and the motion task MT1 is continued immediately after the wait_for_cond(cond_1) with the positioning command pos2(). The return from the "synchronously clocked system level 2" to the positioning command pos2(), i.e. to the "sequential user level", is indicated by the dashed arrow.

[0034]   The fact that, when the condition of the wait_for_condition command has not been satisfied, the checking for the condition takes place in a high-priority "synchronously clocked system level" and, when the condition has been satisfied, the interrupted "motion task" is continued immediately makes it possible for a user to specify extremely time-critical applications

by simple language means during the programming of sequences of movements. The performance and deterministics are further enhanced by only inserting and considering currently applicable conditions when checking the conditions in the respective high-priority "synchronously clocked system levels".

[0035] The mechanism described here also does not require an explicit event handler. The great advantage from the user viewpoint is consequently that the user can now formulate high-priority events in a sequential running program on a relatively low priority level of a "motion task" in his program flow with the aid of program constructs, and does not have to change into another program which he then has to project by means of other mechanisms (for example manually or under interrupt control) onto a synchronous user task, but instead has the possibility in a closed user program of formulating this "waiting for high-priority event" and "high-priority reaction" cycle for this event in a program on a closed basis.

[0036] The conditions which are inquired in a wait_for_condition command can be formulated very flexibly and elegantly by the user. For instance, for formulating these conditions, the user can use program variables from a user program or internal variables of the controller, or he can also reference process signals. These variables may then be combined logically, arithmetically or by any desired functions in terms of their content, to formulate a condition from them. In addition to the high-priority inquiries as to whether the condition is satisfied, it is also conceivable that, if the condition is satisfied, a program code belonging to it, i.e. an underlying response, which is user-programmable, is also executed with high priority. And that the return to the low-priority level only takes place after execution of this program code.

[0037] The representation according to Figure 7 shows an extended exemplary embodiment of the use and mechanism of the wait_for_condition command, in the running level model of the industrial controller according to the invention. The wait_for_condition command (in figure 7 likewise represented as wait_for_cond()) is used by way of example in this representation in the "sequential user level". The wait_for_condition command is used in the "motion tasks" MT3 and MT4 created by the user, which are a component part of the "sequential user level". The "motion tasks" MT3 and MT4 are in a round-robin cycle, represented by the arrow from MT3 to MT4 and by the angled-away arrow from MT4 to MT3. The three points inside this arrow indicate that there may be still further "motion tasks" in the round-robin cycle. The "motion task" MT3 contains the wait_for_condition command "wait_for_cond(cond_3)", the "motion task" MT4 contains the wait_for_condition command "wait_for_cond(cond_4)". Three points used in each case within MT3 and MT4 indicate that, in addition to the two wait_for_condition commands and the positioning commands pos4() to pos8(), still further commands may be contained in the "motion tasks". The programming-language constructs "wait_for_cond()" and "end_wait_for_cond" have the effect of bracketing a program sequence in the "motion tasks". In the "motion task" MT3, the commands pos5() and pos6() are bracketed in this way. The use of "wait_for_cond()" and "end_wait_for_cond" is also indicated in the "motion task" MT4. It is schematically indicated by 3 points in each case in the "motion task" MT4 that further instructions may be present before, within and after the "wait_for_cond() - end_wait_for_cond" construct.

[0038] The running level model, represented by way of example in Figure 7, of a runtime system for an industrial controller comprises, as in Figure 6, the following levels (enumerated

from the lowest to the highest priority): "cyclical background level", "sequential user level" (the tasks of these two levels have the same priority, represented by the dashed line between these levels), "event-controlled user level", "time-controlled user level", "user level for system exceptions", "synchronously clocked user level 2", "synchronously clocked user level 1", "synchronously clocked system level 2" and, as the highest-priority level, a "synchronously clocked system level 1".

[0039]   In Figure 7, the operating mode of the wait_for_condition command with an associated program sequence is shown by way of example as "wait_for_cond(cond_3)" from the "motion task" MT3. The checking of the condition cond_3 and the processing of the associated program sequence (bracketed between "wait_for_cond(cond_3)" and "end_wait_for_cond") take place in this case on a higher-priority level of the running level model. The program sequence belonging to "wait_for_cond(cond_3)" is formed by the sequence of the commands pos5() and pos6().

[0040]   If the "motion task" MT3 is next in turn in the round-robin cycle, the commands of the "motion task" MT3 are serviced until the time slice has elapsed, or an interruption occurs. If this is the case, the "motion task" MT4 is serviced as the next task in the cycle, etc. If the "wait_for_cond(cond_3)" command is processed in the "motion task" MT3, the condition cond_3 is checked. If cond_3 = true, that is to say is satisfied, the normal program sequence is continued, i.e. the command pos5() is executed next and, if appropriate, further commands present in MT3 are successively processed, until control is passed to the next motion task.

[0041]   If the condition cond_3 = false, that is to say is not satisfied, the "motion task" MT3 is immediately interrupted and MT4 is serviced in the round-robin cycle. The condition cond_3

and the commands pos5() and pos6() (as the associated program sequence) are processed in the

priority of the "synchronously clocked system level 2" (indicated by the solid angled-away

arrow, starting from the parenthesis which expresses the unified nature of

wait_for_cond(cond_3), end_wait_for_cond and the associated program sequence, up to the

"synchronously clocked system level 2"). cond_3 is checked in the clock cycle of this system

level to ascertain whether it has been satisfied. If cond_3 has been satisfied, the associated

program sequence (here: the sequence of the commands pos5() and pos6()) is processed with the

priority of the "synchronously clocked system level 2". The return from the "synchronously

clocked system level 2" to the positioning command pos7(), i.e. to the "sequential user level", is

indicated by the dashed arrow.

[0042]   The fact that, when the condition of the wait_for_condition command has not been

satisfied, the checking for the condition takes place in a high-priority "synchronously clocked

system level" and, when the condition has been satisfied, an associated program sequence which

can be created by the user is executed immediately on this high-priority system level makes it

possible for even extremely time-critical applications to be specified and carried out by simple

language means.

[0043]   One possible application is printed mark synchronization. The aim here is to detect a

printed mark on a material with high priority. When this printed mark is detected, typically an

actual value is captured ("latching" for example of a position or sensor actual value). On the

basis of this captured actual value, a correction value is calculated and impressed on the system

as a superposed movement. The process of actual value detection, correction value calculation

and implementation of the superposed movement must take place in a deterministic time period. Therefore, this process must take place with high priority.

[0044] A further application is the "rapid start of movement". Here, the aim is to detect, for example, an edge change very quickly and then begin a start of movement (for example positioning movement) immediately thereafter. The deterministics of detecting an event and triggering consequent actions are decisive for the productivity of a machine. In the case of production machines, such cyclical processes must take place in a deterministic time, for example <100 ms or <50 ms. When processing the tasks on a normal background level, these deterministics cannot be guaranteed. The mechanism described is particularly suitable for use in the case of machines which have periodic machine cycles.

[0045] The performance is further enhanced by only inserting and considering currently applicable conditions when checking the conditions in the respective high-priority "synchronously clocked system levels".

[0046] As already mentioned in Figure 6, the mechanism described here does not require an explicit event handler. The great advantage from the user viewpoint is consequently that the user can now formulate high-priority events in a sequential running program on a relatively low priority level of a "motion task" in his program flow with the aid of program constructs, and does not have to change into another program which he then has to project by means of other mechanisms (for example manually or under interrupt control) onto a synchronous user task, but instead has the possibility in a closed user program of formulating this "waiting for high-priority event" and "high-priority reaction" cycle for this event in a program on a closed basis.

possibilities for how the basic clock is obtained for the industrial controller. Figure 10 shows by way of example a communication topology into which the controller S is integrated. The controller S is represented by a square. The controller S is connected by a connection line A2 to the bus B1, to which the external device EG is attached via a connection line A1. The connection to the technical process P2 takes place via the bus B2. The technical process P2 is represented at the lower edge of the figure by a rectangle. The controller S is connected via the connection line A3 to the bus B2, which in turn establishes the connection to the technical process P2 via the connection line A4.

[0052]   The generation for the basic clock of the controller S can take place from different clock sources. For example, from an internal clock source, represented by the internal timer T2 of the controller S or else by an external clock source, such as for example the timer T1, which belongs to the external device EG. The basic clock of a communication medium may also serve, however, as an external clock source. If the bus B2 is realized for example by an equidistant Profibus, the clock for the controller can be obtained from the basic clock of this bus. This is represented in figure 10 by the timer T3 being positioned directly on the connection line A3, and this connection line A3 establishes the connection to the bus B2. The controller is consequently attached to the bus as a slave and can use the bus clock directly. Furthermore, a clock generator TG which is integrated in the technical process P2 may serve as an external clock source. A clock generator TG in a technical process may be, for example, the operating cycle of a production machine or packaging machine. In the representation according to figure 10, bus connections are represented by way of example as communication media. However, ring, star or other types of connection may also be chosen as communication media, as well as wireless

connections. The basic clock mentioned above can then be derived from these connection systems.